C++学习笔记1

2022年3月19日 20:30

- 开始
  - c++的输入输出
    - c++没有标准的输入输出语句，只有控制流
    - 控制流的储存文件：iostream
    - 输入格式
      - std：：cin >> a
      - 目标：向控制流中写入一个数字赋予a
      - 注意
        - 对于单个的变量，例如i，j，std：：cin在遇到空格和换行符的时候会默认跳过，去读取下一个
        - 对于一个数组类变量，std：：cin输入的时候会直接读取到一个字符串，碰到任何非法字符都会停止（包括括号）。
        - >> 是会过滤掉不可见字符（如 空格 回车，TAB 等），不想略过空白字符，那就使用 noskipws 流控制：cin>>noskipws>>str; 此时输入：空格 test，会输出空格
    - 输出格式
      - std：：cout << "hello world" << std ：：endl;
      - 注意：
        - 在这里“"代表着输入输出的文本文字，注意在其中对于文本中需要空格的地方的控制
        - 注意输出的时候使用std：：cout后要及时用std：：endl刷新缓存区
    - 文件结束符
      - 当从键盘向程序输入数据时候，对于如何指出文件结束是非常必要的
        - windows：ctrl+z代表结束
        - unix或者macos：ctrl+d；代表结束
      - 文件结束符的输入非常重要，因为std：：cin>>输入流的操作会省略部分的运算符，如果不放在循环处就会导致循环出现问题
  - c++的循环，判断
    - 循环
      - 循环遵照c的原则，同样可以正常使用
    - 判断
      - 大部分遵照c的原则，同样可以正常使用
      - 特殊
        - 输入流可以作为判断条件
          - 例如：if（std：：cin>>i）；
          - 在这期间程序同时做了两件事：读取了一个数，判断是否正确，

如果遇到数字整个程序会返回1，继续执行下面的语句，否则返回0，不去执行下面的语句。

- 类
  - 一般的，要使用类，我们需要使用头文件来访问为自己应用设定的类，习惯上头文件根据其中定义的类的名字来命名
  - 要调用一个类，就要#include “xxx。h”
    - 我们一般用。h作为头文件的后缀
  - 类定义了行为：当你使用了一个类的时候，其加减乘除操作就已经被确定且不能被更改了，这点要注意
  - 成员函数
    - 方法：使用。（点运算符）来表达需要的item1对象的isbn成员
      - 例如：item。isbn
      - 其中
        - 点运算符只能用于类类型的对象（xx。h）
    - 调用函数的方法：需要使用调用运算符（）来调用一个函数，调用运算符里面是一对圆括号，里面放置实参列表，有些成员函数例如isbn不接收函数
    - 举例
      - item1.isbn（）
      - 调用了item1对象的isbn函数，因其不需要输入值，所以直接返回item1中保存的内容

- 基础
  - 变量的定义
    - 包括基本数据类型
      - 如int，double都叫基本数据类型
    - 和声明符
      - 如*，&，等等，用来修饰定义的变量类型
      - 如int * p代表定义了一个int类型，*代表他是一个指针
      - 注意可以理解为声明符一般只有一个（实际上由两个），如int **p申明了一个指针类型 （int *）的指针
    - 一般变量的定义都包括基本数据类型和声明符，其中声明符是可有可无的东西，使用特殊功能的时候需要加上*，&等
  - 选择类型的一些准则
    - 当明知数值不可能为负数时候，选用无符号类型。
    - 使用int执行整数运算。如果你的数值超过了int的表示范围，选用long long
    - 在算术表达式中不要使用bool或char类型，只有在存放字符或者布尔值时候才使用它们。
    - 如果你需要使用一个不大的整数，那么明确指定它的类型是signed char或者unsigned char。
    - 执行浮点数的时候选用double

        - 切勿混用带符号类型和无符号类型
    - 字符串，数组
        - ‘’ 由单引号括起来的一个字符称为char型字面值
        - “”双引号括起来的零个或多个字符则构成字符串型字面值
    - 字面值
        - 通过添加字面值，可以改变整型，浮点型和字符型的默认类型
    - 初始值
        - 当对象在创建时候获得了一个特定的值，我们说这个对象被初始化了。
        - 注意，初始化不是赋值，初始化的含义是创建变量时候赋予其一个初始值，而赋值的含义是把对象当前值擦掉，而以一个新值来代替
    - 初始化的四种方法
        - int a=0;
        - int a={0};
            - 作为c++11的新标准，用花括号来初始化一个数字得到了全面的应用
            - 这种初始化方法叫做链表初始化。
        - int a（0）；
        - int a{0};
        - 默认初始化
            - 在函数体外部的变量如果没有指定初始值，则变量被默认初始化，这由他的变量类型决定，也与 它所在的位置有关
            - 在函数体内部的变量是不被初始化的，一个未被初始化的内置类型的变量的值是未定义的，如果试图拷贝或者访问会发生错误
            - 一些类要求每个对象都要显式初始化，此时如果创建理论一个该类对象未对其初始化的操作则引发错误
        - 注意
            - 括号/花括号里面可以放字母也可以放数字，都是没问题的。只要类型相同
            - 带花括号会检查转换项，检测类型是否符合，如果不符合将要报错
            - 要初始化每一个内置类型的变量，保证后续程序的安全。
            - 初始化是初始化，赋值是赋值，初始化的同时不能再被赋值
    - 声明与定义
        - 声明：使名字被程序知道，一个文件如果想使用别处定义的名字则必须包含对那个名字的声明。
            - extern int i；----》声明i而非定义i
        - 定义：负责创建于名字关联的实体，申请储存空间，也可能为变量赋一个初始值
            - int j；-------》声明且定义了i
        - 注意
            - 在函数内部，如果试图初始化一个由extern关键词标记的变量，将引发错误
            - 变量只能定义一次，但是可以被多次声明
    - 复合类型：c++由几种复合类型，包括引用与指针
        - 定义

        - 复合类型是指基于其他类型定义的类型，定义复合类型的变量要复杂很多
        - 一条声明语句由一个基本数据类型和紧随其后的一个声明符列表组成。
        - 每个声明符命名了一个变量并指定该变量为与基本数据类型有关的某种类型
    - 引用（右值引用）与指针
        - 备注：c++11中新增了一种引用，所谓的“右值引用”现在这种引用叫做左值引用
        - 引用即别名，是一个变量另外的名字，使用别名的时候真正的对象是指向被引用的对象
            - int &i = j;
                - 此时，i是j的别名（i是j的小名）
            - 引用本身不是一个对象，所以不能定义引用的引用
                - 此时如果int &a = i就是错的，应为定义了引用的引用
            - 引用的类型必须与其所引用的对象一致，const除外
        - 指针
            - 指针是一个存放地址的一个变量
            - 指针的四种状态
                - 指向一个对象
                - 指向紧邻对象所占空间的下一个位置
                - 空指针，意味着指针没有指向任何对象
                - 无效指针，也就是上述情况的其他值
                - 注意：对于无效指针，编译器不能检查出来，但是其会对程序造成比较大的危害，所以无效指针必须避免，不需要的指针一定要初始化。
            - 指针的赋值
                - 包括cstdlib.h，用预处理使用null赋值
                - 使用nullptr字面值进行赋值
            - 指针的特殊类型空指针
                - void*
                - void*的作用
                    - 作为函数的输入输出
                    - 赋予另外一个void指针
    - const的用法
        - const的2种赋值方法
            - const int i=hanshu（）
            - const int i=5;
            - const int i{5};
            - 注
                - 能对正常值进行初始化的都行，比较特殊的一种就是函数初始化
                - 对函数初始化时运行时候初始化，直接复制是对编译的时候初始化
        - const的多文件用法
            - 某些时候有这样一种const变量，他的初始值不是一个常量表达式，但又确实

有必要在文件间共享。这种情况下，我们不希望编译器为每个文件分别生成独立的变量。相反，我们想让这类const对象像其他（非常量）对象一样工作，也就是说，只在一个文件中定义const，而在多个文件中使用它。

- 如果想在多个文件之间共享const对象，必须在变量的定义之前加extern关键词。
- 对常量的引用
  - 使用方法
    - const int j=2;
    - const int &i=j;
    - 注意：
      - 对const的引用可能引用一个并非const的对象
        - const int类型可以引用int类型的，但是反过来不行
          - 正确
            - int c{5};
            - const int &d=c;
          - 错误
            - const int c{5};
            - int &b=c;
      - 无法通过const引用来修改对象的值，尽管原对象的值可以被修改
        - int c{5};
        - const int &b=c;
        - b=2（这句话会报错）
      - 简而言之，const对应的对象才会有可变和不可变之说，但是给其赋值的变量相等不相等不知道
        - 例如const int a=b，a一定是个常量，但是b是不是常量不知道
- 常量指针
  - 像其他类型一样，指针也可以被定义为常量，必须初始化，初始化完成后不在改变，但是可以通过指针去修改对应地址上的值，只是指针不再变化而已
  - 顶层const和底层const
    - 顶层const：表示指针本身是个常量。
      - 顶层const表示自己是常量，对象是变量，对任何数据类型都适用
      - 顶层const对应的对象是可以更改的
    - 底层const：表示指针所指的对象是一个常量。
      - 底层const对应的对象是不可以更改的（常量）自己可以更改
      - 当执行对象的拷贝操作的时候，拷入和拷出的对象必须具有相同的底层const资格
    - 引用永远都是一个底层const
- constexpr类型的函数（c++11）
  - c++11新标准规定，允许将变量声明为constexpr类型以便由编译器来验证变量的值是否是一个常量表达式

- 注意
  - 声明为constexpr的变量一定是一个常量，而且必须用常量表达式初始化
  - 所谓常量就是拥有一个固定的值的量。
  - 一般来说，如果你认定变量是一个常量表达式，那就把它声明成constexpr类型
  - 指针和constexpr
    - 指针和引用也能定义constexpr，但是他们需要的初始值要为0或者nullptr
    - 必须明确一点，如果在constexpr声明中如果定义了一个指针，限定符constexpr仅仅对指针有效，与指针所指的对象无关
      - const int *p---》 *p被限制
      - constexpr int *p=nullptr————》p被限制
        - 作为一个常量指针，它同样也可以赋予一个常量
- 举例
  - constexpr int mf=20; 20是一个常量表达式
  - constexpr int limit =mf+1; mf+1也是一个常量表达式
  - constexpr int sz=size(); 只有当size是一个constexpr是一个具体的申明的时候，才是一条正确的声明语句

- 注意
  - 通过const定义的任何变量，引用也好，指针也好，都是不可以被修改的，（但是引用不是对象）
  - 判断const和*前后的关系
    - 从右边向左边读，遇到const那么他右边那个就是一个常量
      - int *const p：p是常量，这是一个常量指针
      - const int *p： *p不可以被修改，p可以修改。
  - 对于多函命名的const类型的指针，先和函数读再和指针读
    - const list p
      - p是一个指针类型的指针，p不能动
  - 分清楚”赋值“和”对象“之间的关系
    - const int a=2----》赋值
    - const int b=x---》b，x都是对象，x或许可以更改，但是b一定不能

- 类型别名
  - 定义方法：typedef （与c语言一样）
    - typedef double wages;
      - 其中，基本数据类型是double ， wages作为基本数据类型double的一个别名
    - typedef int* x;
      - 其中，基本数据类型是int，声明符为*，但是x把int *作为一个新的数据类型（指针类型）进行判断

        - 例
            - typedef char * x;
            - const x c----》这是一个char类型的常量指针
            - const x *e----》e是一个 x 类型的指针， *e是一个char类型的常量指针
        - 注意：不能对定义类型进行直接的替换，可能会出现数据类型的错误，必须要先看声明，在结合着一起看
    - 定义方法：using
        - using si=sale——xx；这里代表si是sale——xx的别名
- auto类型
    - auto类型：编译器可以根据量的关系给他自动的分配一个类型
        - 例如，如果a是int类型（int a）
        - 则 auto b=a，此时b为会被编译器赋予int类型
    - auto类型一般会忽略掉顶层const
        - auto会透过现象看本质，直接找到本源那个变量的类型然后赋值
            - const int a=b，auto c=a，则c会被赋予b的变量本质
        - 例如 int *const b=xx，auto c= *b，auto d= b此时c为int类型，d为int *（指针类型），b为底层const不能被修改，故d也不能修改
        - 例如 const int* d=xx，auto c=*d，此时auto为const int类型，c可以被修改的，
    - 如果希望自己的auto类型是一个顶层const 需要
        - const auto x;
    - auto类型同样适用引用
- decltype 类型
    - 作用：希望从表达式的类型推断出要定义的变量的类型。但是不想用该表达式的值初始化变量
    - 格式：decltype（f（））sum=x;
        - 其中sum的类型就是函数f的返回类型
        - 举例
            - const int ci =0 &cj=ci;
            - decltype（ci）x=0---》x为const int
            - decltype （cj）r==x---》r为const int&类型
            - declttype（cj）z----》错的，引用必须初始化
    - decltype的特殊使用方法
        - 如何把一个引用转化为对应的类型
            - decltype（r+0）r为引用，r+0变成一个类型的值而非引用
        - decltype表达式的内容是解引用操作，则decltype将得到引用类型
            - decltype （*p）c-----》c为对应 *p变量类型的引用
        - 对于decltype来说
            - 如果变量名加了一对括号，则得到的类型与不加括号的变量会相同

                - 如果给变量加上了一层或者多层括号，编译器就会把它当成一个表达式，变量是一种可以作为赋值语句左值的特殊表达式
                -
    - auto与decltype的区别
        - 不同
            - auto会忽略顶层const ，而decltype不会
            - auto在创建变量的时候会自动的让auto创建的变量赋值，而decltype只是取其类型
        - 相同
            - auto与decltype都是一个类似于int之类的类型
            - auto与decltype都是用来推断的
    - 自定义数据结构
        - 简单的，我们同样可以使用struct来定义类
        - 注意
            - 一般来说，最好不要把对象的定义和类的定义放在一起，这么做无异于把两种不同实体的定义混在一条语句里面，一会定义类，一会又定义变量，显然这是一种不被建议的行为
            - c++11新标准规定，可以为数据成员提供一个类内初始值。创建对象时，类内初始值将用于初始化数据成员。没有初始化的成员将会被默认初始化。
            - 初始化的时候，只能使用花括号和等号初始化值，不能使用圆括号
        - 头文件与预处理器
            - 头文件
                - 为了确保各个文件中的类的定义一致，类通常被定义在头文件中，而且类所在的头文件的名字应该与类的名字一样。
                - 头文件通常包含哪些只能被定义一次的实体，如类，const constexpr变量
                - 头文件一旦改变，相关的源文件必须重新编译以获取更新过的声明
            - 预处理器
                - 确保头文件多次包含仍能安全工作的常用技术是预处理器，它从c语言继承过来
                - 头文件保护符
                    - 头文件保护符依赖于预处理变量，预处理变量有两种状态，已定义和未定义
                    - #define指令把一个名字设定为预处理变量，另外两个指令则分别检查某个指定的预处理变量是否已经定义
                    - #ifdef当且仅当变量已定义时为真，#ifndef当且仅当变量未定义时候为真。一旦检查结果为真，则执行后续操作直至遇到#endif指令为止
                    - 预处理变量无视c++语言中关于作用域的规则
                    - 预处理变量包括的头文件保护符必须唯一，通常的做法是基于头

文件中类的名字来构建保护符的名字，以确保唯一性。

- ◇ 为了避免与程序中其他实体发生名字冲突，一般把预处理变量名字全部大写。
- ◇ 头文件即使还没有被包括在任何其他头文件中，也应该设置保护符，头文件保护符很简单，只要习惯性加上就可以了，没必要在意你的程序需不需要

- 小结
  - 类型是c++编程的基础
  - 类型规定了其对象的存储要求和所能执行的操作。c++语言提供了一套基础内置类型，如int和char等，这些类型与实现他们的机器硬件密切相关
  - 类型分为非常量和常量，一个常量对象必须初始化，而且一旦初始化其值就不能再改变。
  - 还可以定义复合类型，如指针和引用等。复合类型的定义以其他类型为基础
  - c++语言允许用户以类的形式自定义类型，c++库通过类提供了一套高级抽象类型，如输入输出string等。